Combining Branch Predictors

Prof. Naga Kandasamy
ECE Department
Drexel University

May 5, 2016

These notes are derived from:

- S. McFarling, *Combining Branch Predictors*, WRL Technical Note TN-36, Western Research Laboratory, June 1993.

We discuss the following methods of dynamic branch prediction:

- *Bimodal branch predictor* makes a prediction based on the direction the branch went the last few times it was executed.
- *Local branch predictor* considers the history of each branch independently and takes advantage of repetitive patterns.
- *Global branch predictor* uses the combined history of all recent branches in making a prediction.

Each of these different branch prediction strategies have distinct advantages. The bimodal technique works well when each branch is strongly biased in a particular direction. The local technique works well for branches with simple repetitive patterns. The global technique works particularly well when the direction taken by sequentially executed branches is highly correlated.

1 Bimodal Branch Prediction

The behavior of typical branches is far from random. Most branches are either usually taken or usually not taken. Bimodal branch prediction takes advantage of this bimodal distribution of branch behavior and attempts to distinguish usually taken from usually not taken branches. A simple approach to achieving this is to maintain a table of counters indexed by the low order address bits in the program counter (see Fig. 1). Each counter is two bits long. For each taken branch, the appropriate counter is incremented. Likewise for each not-taken branch, the appropriate counter is decremented. In addition, the counter is saturating. In other words, the counter is not decremented past zero, nor is it incremented past three. The most significant bit determines the prediction. Repeatedly taken branches will be predicted to be taken, and repeatedly not-taken branches will be predicted to be not-taken. By using a 2-bit counter, the predictor can tolerate a branch going an unusual direction one time and keep predicting the usual branch direction.

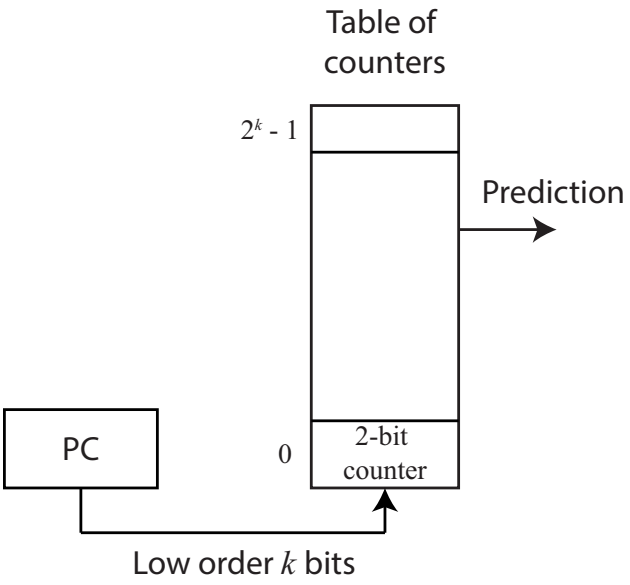

Figure 1: Structure of the bimodal predictor.

For large counter tables, each branch will map to a unique counter. For smaller tables, multiple branches may share the same counter, resulting in degraded prediction accuracy. One alternate implementation is to store a tag with each counter and use a set-associative lookup to match counters with branches. For a fixed number of counters, a set-associative table has better performance. However, once the size of tags is accounted for, a simple array of counters often has better performance for a given predictor size.

2 Local Branch Prediction

Consider the following loop example:

```
for (i = 1; i <= 4; i++){

    // Loop body

}  // Branch test is done here
```

One way to improve on bimodal prediction is to recognize that many branches execute repetitive patterns. In the above example, if the loop test is done at the end of the body, the corresponding branch will execute the pattern $(1110)^n$, where 1 and 0 represent taken and not taken respectively, and n is the number of times the loop is executed. Clearly, if we knew the direction this branch had gone on the previous three executions, then we could always be able to predict the next branch direction.

To exploit the knowledge of repetitive patterns executed by a branch, we use a branch predictor with two tables (see Fig. 2). The first table records the history of recent branches. We will assume that it is simply an array indexed by the low-order bits of the branch address. Each history table entry records the direction taken by the most recent n branches whose addresses map to this entry, where n is the length of the entry in bits. The second table is an array of 2-bit counters identical

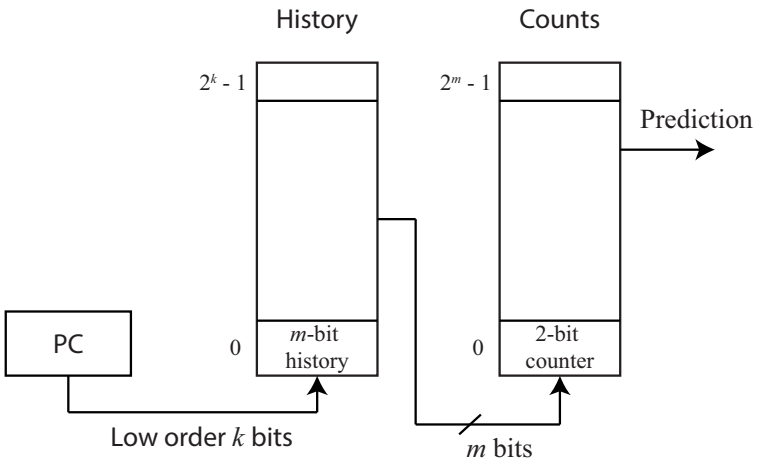

Figure 2: Structure of the local history predictor.

to those used for bimodal branch prediction. However, here they are indexed by the branch history stored in the first table.

Consider again the simple loop example above. Assume that this is the only branch in the program. In this case, there will be a history table entry that stores the history of this branch only and the counter table will reflect solely the behavior of this branch. With 3 bits of history and 2^3 counters, the local branch predictor will be able determine the current iteration and always make the correct prediction after some initial settling of the counter values. If there are more branches in the program, a local predictor can suffer from two kinds of contention. First, the branch history may reflect a mix of histories of all the branches that map to each history entry. Second, since there is only one counter array for all branches, there may be conflict between patterns. For example, if there is another branch that typically executes the pattern $(0110)^n$ instead of $(1110)^n$, there will be contention when the branch history is (110). However, with 4 bits of history and 2^4 counters, this contention can be avoided. Note however, that if the first pattern is executed a large number of times followed by a large number of executions of the second pattern, then only 3 bits of history are needed since the counters dynamically adjust to the more recent patterns.

3 Global Branch Prediction

In the local branch prediction scheme, the only patterns considered are those of the current branch. Once can also take advantage of the behavior of other recent branches to make a prediction. A single shift register called GR records the direction taken by the most recent n conditional branches. Since the branch history is global to all branches, this strategy is called global branch prediction. This method is able to take advantage of two types of patterns:

- The direction taken by the current branch may depend strongly on other recent branches. Consider the example below:

  ```
  b1: if (x < 1) a = 0;
  b2: if (x > 1) a = 1;
  ```

  If the two branches are labeled `b1` and `b2`, respectively, using global history prediction, we

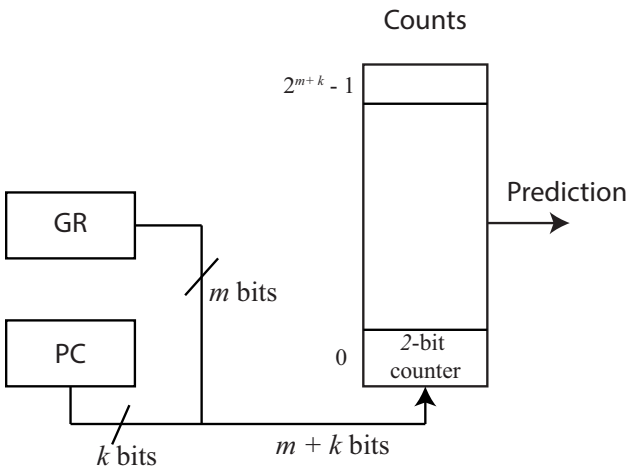

Figure 3: Structure of the global history predictor with index selection or *gselect*.

are able to base the prediction for `b2` based on the direction taken by `b1`. If $x < 1$, we know that `b2` will not be taken. If, however, $x \geq 1$ then we do not know conclusively which way `b2` will be taken, but the probability may well be skewed one direction or the other. If so, we should be able to make a better prediction than if we had no information about the value of x.

- A second way that global branch prediction can be effective is by duplicating the behavior of local branch prediction. This can occur when the global history includes all the local history needed to make an accurate prediction. Consider the example:

```
for(i = 0; i < 100; i++){
    for(j = 0; j < 3; j++){

    }
}
```

After the initial startup time, the conditional branches have the following behavior, assuming GR is shifted to the left:

Loop test	Value	GR	Branch outcome
$j < 3$	$j = 1$	1101	Taken
$j < 3$	$j = 2$	1011	Taken
$j < 3$	$j = 3$	0111	Not taken
$i < 100$		1110	Usually taken

A more efficient global prediction scheme uses both the branch address and the global history (see Fig. 3) to index into the Counts table. This method is called global prediction with index selection or *gselect*. Here the counter table is indexed with a concatenation of global history and branch address bits.

Finally, we discuss an improvement over *gselect* called global history with index sharing or *gshare*. Consider the following simple example where there are only two branches and each branch has

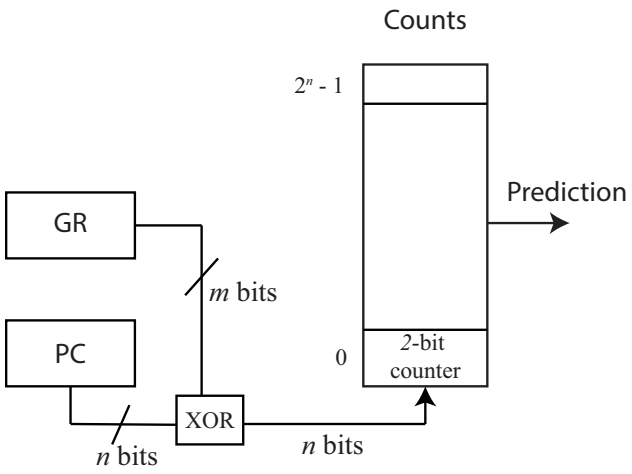

Figure 4: Structure of the global history predictor with index sharing or *gshare*.

only two common global histories:

Branch address	Global history	*gselect* 4/4	*gshare* 8/8
00000000	00000001	00000001	00000001
00000000	00000000	00000000	00000000
11111111	00000000	11110000	11111111
11111111	10000000	11110000	01111111

Strategy *gselect* 4/4 concatenates the low order 4 bits of both the branch address and the global history. We will call the strategy of exclusive ORing branch address and global history *gshare*. Strategy *gshare* 8/8 uses the bit-wise exclusive OR of all 8 bits of both the branch address and the global history. Comparing *gshare* 8/8 and *gselect* 4/4 shows that only *gshare* is able to separate all four cases. The *gselect* predictor cannot take advantage of the distinguishing history in the upper four bits. Figure 4 shows the *gshare* predictor structure.

As with *gselect*, we can choose to use fewer global history bits than branch address bits. In this case, the global history bits are exclusive ORed with the higher order address bits. Typically, the higher order address bits will be more sparse than the lower order bits.

4 Correlating Branch Predictors

Consider the following code fragment from the SPEC92 benchmark eqntott that exhibits bad branch prediction behavior:

```
if (aa == 2) aa = 0;
if (bb == 2) bb = 0;
if(aa != bb){...}
```

The MIPS code that would be generated assuming `aa` and `bb` are assigned to registers R1 and R2 is as follows:

```
     ADD R3, R1, #-2
     BNEZ R3, L1             ; branch b1 (aa != 2)
     ADD R1, $zero, $zero    ; aa = 0
L1:  ADD R3, R2, #-2
     BNEZ R3 L2              ; branch b2 (bb != 2)
     ADD R2, $zero, $zero    ; bb = 0
L2:  SUB R3, R1, R2          ; R3 = aa - bb
     BEQZ R3, L3             ; branch b3 (aa == bb)
```

Observe that the behavior of branch b3 is correlated with the behavior of branches b1 and b2 in that if b1 and b2 are both not taken, then b3 will be taken since both `aa` and `b` are assigned 0 and are clearly equal. Branch predictors that use the behavior of other branches to make a prediction are called correlating predictors or two-level predictors. For example, a (2, 2) predictor uses the behavior of the last two branches to choose from a set of four 2-bit branch predictors in predicting a particular branch. In general, an (m, n) predictor uses the behavior of the last m branches to choose from 2^m branch predictors, each of which is an n-bit predictor for a single branch.

Very little extra hardware is needed to implement a correlating predictor starting with an existing 2-bit predictor. The global history of the most recent m branches can be recorded in an m-bit shift register, where each bit records the branch outcome—taken or not taken. The branch prediction buffer can then be indexed using a concatenation of the low-order bits of the branch address with the m-bit global history. For example, in a $(2, 2)$ buffer with 64 entries, the 4 low-order bits of the branch address and the 2 global bits representing the behavior of the two most recently executed branches form a 6-bit index that can be used to index the 64 counters.

5 Combining Branch Predictors

The different branch prediction schemes presented above have different advantages. A natural question is whether the different advantages can be combined in a new branch prediction method with better prediction accuracy. One such method is shown in Fig. 5. This combined predictor contains two predictors P1 and P2 that could be one of the predictors discussed in the previous sections. In addition, the combined predictor contains an additional counter array which serves to select the best predictor to use. As before, we will use 2-bit up/down saturating counters. Each counter keeps track of which predictor is more accurate for the branches that share that counter. Specifically, using the notation P1c and P2c to denote whether predictors P1 and P2 are correct respectively, the counter is incremented or decremented by P1c-P2c as shown in the table below.

P1c	P2c	P1c - P2c	
0	0	0	No change
0	1	-1	Decrement counter
1	0	1	Increment counter
1	1	0	No change

One combination of branch predictors that is useful is bimodal/gshare. In this combination, global information can be used if it is worthwhile, otherwise the usual branch direction as predicted by the

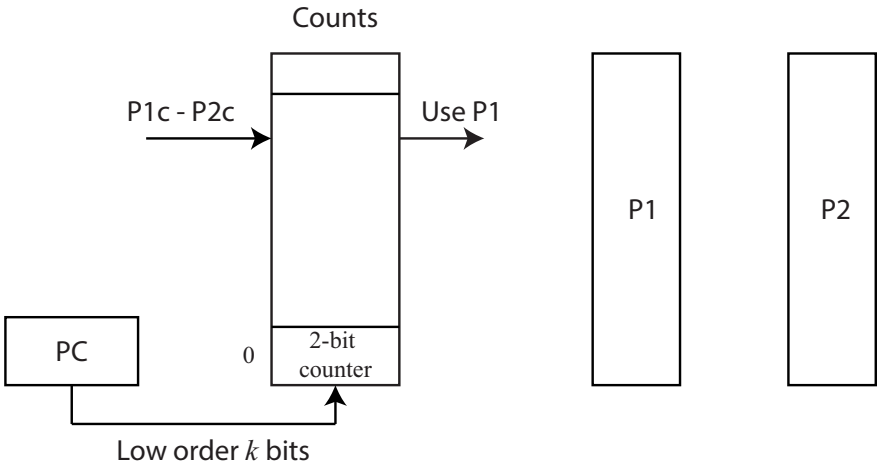

Figure 5: Combined predictor structure.

bimodal scheme can be used. Here, we assume gshare uses the same number of history and address bits. This assumption maximizes the amount of global information. Diluting the branch address information is less of a concern because the bimodal prediction can always be used. Similarly, gshare performs significantly better here than gselect since it usesmore global information.